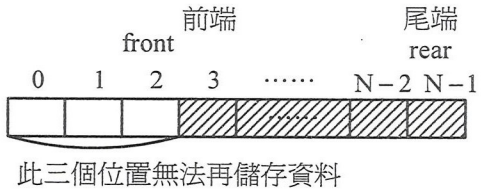
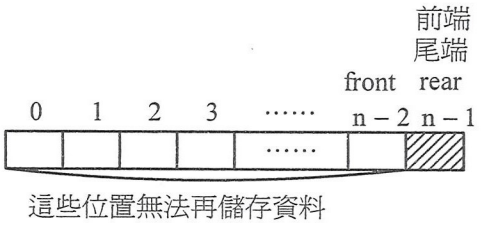
依照上述加入與刪除資料的方式操作之後，可能出現圖 4-5 的情形：



(a) 浪費了 3 個空間



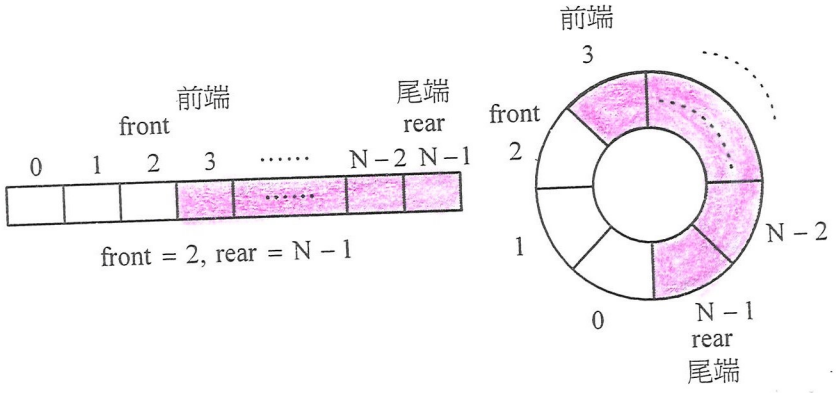
(b) 浪費了 n−1 個空間

圖 4-5　佇列空間無法再利用的情形

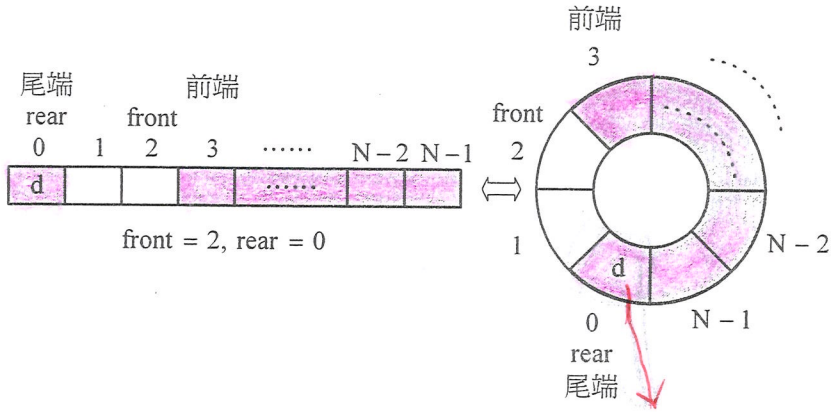
因爲資料的加入是置於目前尾端的下一個位置，而圖 4-5 中目前尾端已無下一個空間，那麼要進入佇列中排隊的資料豈不是不能加入了？這似乎像是有人霸佔著買票的入口處，不讓其他人進入排隊一般，而前面空著的空間就浪費了！要如何改善這問題呢？

要改善以上所提的問題，有一個方法是當有資料要加入時，若尾端之後已無空間，但佇列前端尚有空間，則將所有資料往前移動，就如同請排隊中的人往前挪一般，但此方法相當費時，當前一個人尚未往前挪動之前，後一個是不可能向前走的，因此若經常作資料的移動將減低系統執行的效率。

另一個改善的方法是將佇列視爲一個環 (circular)，如圖 4-6(a)，將整個佇列想像成頭、尾相接的環狀，當尾端位置爲 N−1 時，此時若要加入資料 d ，而位置 0 處的空間空著時，自然就將資料 d 加入此處，且尾端也跟著移到了位置 0 處，如圖 4-6(b) 所示。



(a) 佇列視爲一個環 (circular)



(b) 加入資料 d 於位置 0 處

圖 4-6　環狀佇列

因此要加入新資料 d 時，若 rear = N − 1，則 rear 改爲 0，否則 rear 加一。程式如下：

```
if (rear==N-1)
    rear=0;
else
    rear++;
```

上述程式可改爲：

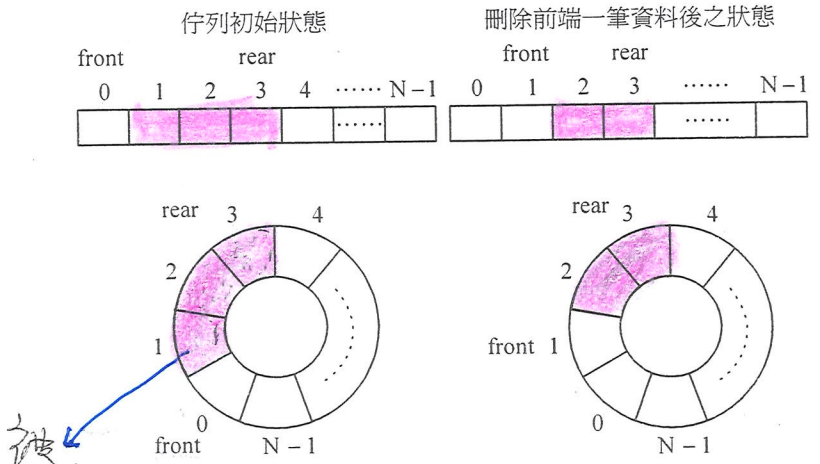
```
(rear==N-1)?rear=0:rear++;
```

或者改爲：

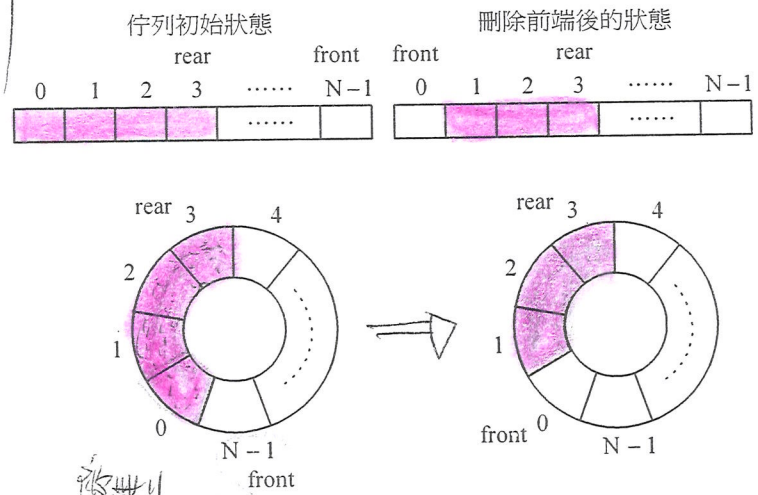
```
rear=(rear+1)%N;
```

其中 % 表示求餘數的意思。

同樣的，刪除環狀佇列的前端資料， front 必須後移一格，即當 front = N − 1 時，則 front 後移至 0 處，否則後移至 front + 1 處，如圖 4-7 所示。



(a) 當 front ≠ N − 1時，front 改爲 front + 1



(b) 當 front = N − 1時，front 改爲 0

圖 4-7　刪除環狀佇列前端資料。

所以刪除前端資料時， front 後移一格與 rear 後移一格的情形一樣，故程式可寫爲：

```
if (front==N-1)
   front=0;
else
   front++;
```

上述程式可改爲：

```
(front==N-1)?front=0:front++;
```

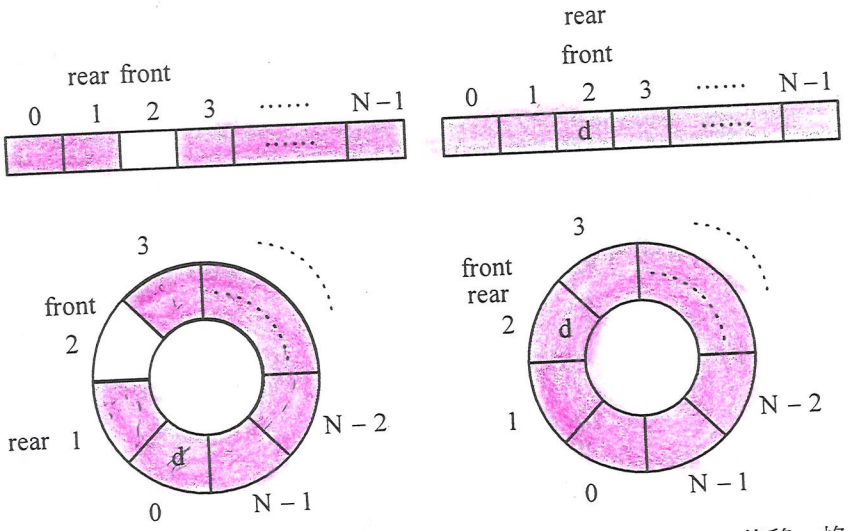
或者：

```
front=(front+1)%N;
```

其中 % 表示求餘數的意思。

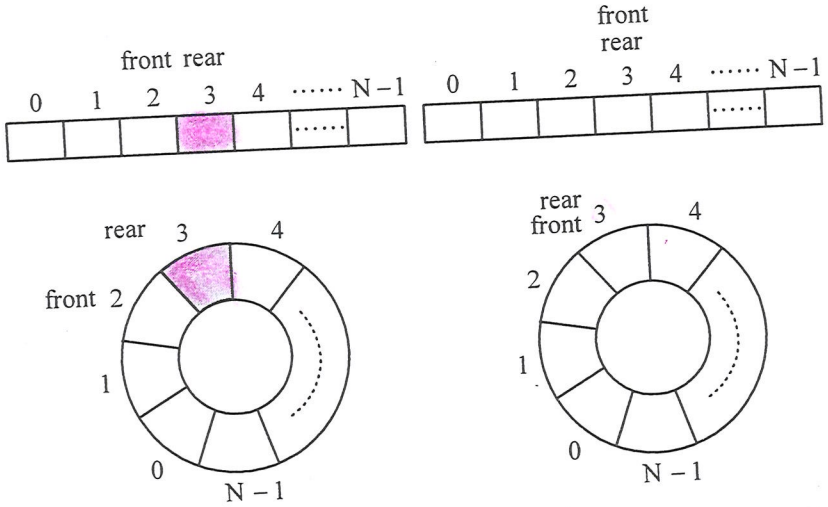
當佇列被視爲環狀後，加入新資料時，仍必須考慮環狀佇列是否滿了，不能再加入。由圖 4-8 得知，加入資料 d 於僅剩的空間後，環狀佇列已滿，不能再加入資料，此時 front = rear 。

刪除環狀佇列資料時，同樣必須考慮是否爲空環狀佇列，已無資料可刪除。由圖 4-9 得知，當刪除僅存的唯一資料後，環狀佇列已空，不能再刪除資料，此時 front = rear ，這也符合 front 與 rear 變數的初值設定皆設爲 −1 時，環狀佇列爲空佇列的條件。



(a) 僅剩一個空間　(b) 加入資料 d 後，rear 後移一格

圖 4-8　環狀佇列已滿⇔ front = rear



(a) 僅剩一筆資料　(b) 刪除資料後，front 後移一格

圖 4-9　空環狀佇列⇔ front = rear

回顧前面的敘述，由圖 4-8 與 4-9 得知：

環狀佇列已滿⇔ （ front == rear ） 成立

環狀佇列已空⇔（ front == rear ） 成立

則當 front = rear 情形出現時，究竟環狀佇列是空的還滿的已無法分辨！爲了解決這個問題，可以考慮當僅剩最後一個空間時，不允許再加入資料視爲環狀佇列已滿，此時 rear 的下一個位置即爲 front ，亦即

```
(rear+1)%N=front
```

如圖 4-10 所示，雖然浪費一個空間，但可避免造成無法分辨空環狀佇或是滿的環狀佇列的困擾。

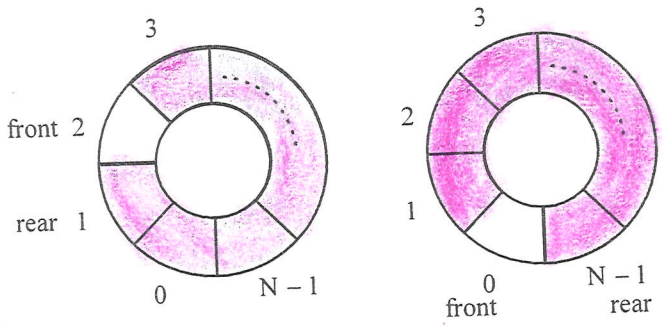
有另一種解決方式是另外設一個變數，例如 tag 來分辨，當加入資料後若佇列已滿則設 tag = 1，當刪除資料後佇列已空，則設 tag = 0，那麼當發生 front = rear 成立時，先檢查 tag 之值，即可知是滿的還是空的，但這方法會降低系統效率，因系統將會多花一些時間去檢測 tag 之值。

有一個比喻可以幫助釐清此觀念。有一隻狗將會有以下的反應：

吃飽了⇔叫三聲，“汪汪汪”！

肚子餓⇔叫三聲，“汪汪汪”！

那麼牠的主人聽見小狗叫三聲，將無法得知是吃飽了，還是肚子還餓著，如果訓練這隻狗狗懂得叫三聲後，再去咬代表飽或餓的牌子，雖然解決了分辨的問題，不如訓練牠吃飽了與肚子餓時，分別叫兩聲與三聲，這樣更快速就能分辨不同的情況了！



(a) 此時 (rear + 1)%N = front (b) 此時 (rear + 1)%N = front

圖 4-10 僅剩一個空間時，視為環狀佇列已滿。

因此目前環狀佇列空或是滿的條件爲：

環狀佇列已滿⇔ （ front == (rear + 1)%N ） 成立

環狀佇列已空 ⇔ （ front == rear ） 成立

所以加入資料 d 於環狀佇列的 add ( ) 函數可簡述如下：

1. 檢查環狀佇列是否滿了，若是滿了則加入失敗。
2. 否則將資料由尾端加入，即 rear 向後移動一格新資料加入目前 rear 位置內。

因此 add ( ) 函數之程式如下：

```
void add(int d)
{
  if(front==(rear+1)%N){
    printf("環狀佇列滿了\n");
    exit(1);                          /*加入失敗，結束程式之執行*/
  }
  rear=(rear+1)%N;
```

```
    queue[rear]=d;
}
```

P.5

刪除環狀佇列的前端資料之 delete ( ) 函數可簡述如下：

1. 檢查環狀佇列是否空了，若是空了則刪除失敗。
2. 否則將資料由前端取出，即 front 向後移動一格，取出目前 front 位置之資料。

（ 注意： front 變數表示真正前端元素的前一個位置 ）

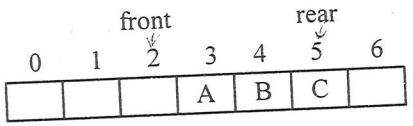
delete ( ) 函數之程式如下：

```
int delete ( )
{
  if (front==rear) {
    printf ("環狀佇列已空\n");
    exit (1);                          /*刪除失敗，結束程式之執行*/
  }
  front=(front+1)%N;
  return (queue [front]);
}
```

陣列製作佇列要點：

1. 佇列視爲一個環，即環狀佇列。
2. front 永遠表示真正前端的前一個位置， rear 永遠表示真正尾端的位置。
3. 當僅剩最後一個空間時，視爲環狀佇列已滿，不能再加入資料。

例 **1** 環狀佇列如下：



front = 2
rear = 5

執行下列動作後，寫出 front 及 rear 的值，並畫出佇列的狀態。

(1) 刪除　front = 2 ，rear = 2 ，
取出內容 = A 。

(2) 加入 D　front = 2 ，rear = 6 ，

(3) 加入 E　front = 2 ，rear = 0 ，

(4) 刪除　front = 3 ，rear = 0 ，
取出內容 = B 。

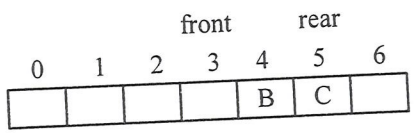
解： (1) 刪除時由前端取資料，即

(a) front 後移一格， rear 不變。

(b) 取出目前 front 位置之資料。

故 front = 3 ，rear = 5 ，

取出內容 = A 。



(2) 加入資料時由尾端加入，即

(a) rear 後移一格， front 不變。